Submitted to 1995 Asian Computing Science Conference #12

New Approaches in Randomized Preprocessing for Motion Planning

Sumanta Guha Rama Devi Puvvada Deepti Suri Ichiro Suzuki*
Electrical Engineering & Computer Science Department
University of Wisconsin-Milwaukee
Milwaukee, WI 53201, USA
guha|puvvada|deepti|suzuki@cs.uwm.edu

Abstract

A powerful new method based on randomization has emerged recently with great success in robot motion planning. Firstly, this consists of preprocessing, in which valid configurations (nodes) of the robot are generated randomly and a local planner is used to join some pairs of nodes, to form a configuration graph. Subsequently, path planning is executed by connecting the given initial and final configurations of the robot to two nodes of the graph, and finding a path joining these two nodes. Our contributions in this paper are two: (a) We describe and implement a new heuristic for node generation, termed *cell division*, based on iterative, but controlled, splitting of the configuration space into smaller cells. The heuristic tends to generate more nodes in "harder" regions of the configuration space, thus allowing the planner a greater choice of nodes in such regions during path planning. (b) We next describe and implement an entirely different approach in randomized preprocessing, termed *ray shooting*, in which we randomly generate rays (instead of configurations) in the configuration space along which we intend to transport the robot, and join some pairs of rays (viewed as nodes) using a local planner, to form a segment graph. Path planning then proceeds as before. Our implementation verifies that the planner performs efficiently even with minimal preprocessing, since each node in the graph represents a track that may transport the robot a relatively large distance through the configuration space.

Keywords: Motion planning, randomization, heuristic
Track: Algorithms

1 Introduction

Motion planning is one of the most active areas of robotics research. It has been studied extensively over the last two decades. Motion planning involves finding a feasible (continuous and collision-free) path for a robot from a start configuration s to a goal configuration g, in some configuration space C whose dimension is determined by the

* *Communicating author*. This author's work was supported in part by the National Science Foundation under grant IRI-9307506, the Office of Naval Research under grant N00014-94-1-0284, and an endowed chair (KIFUKOZA) supported by Hitachi Ltd. at Faculty of Engineering Science, Osaka University.

number of degrees of freedom of the robot. All the configurations on a path (as well as, of course, s and g) should belong to the subspace C_f of free configurations where the robot does not intersect an obstacle. See [3] for a survey of motion planning.

The classical approaches to motion planning can generally be divided into the following classes: *skeleton (roadmap), cell decomposition* and *potential field* approaches.

In the skeleton approach [3, 7, 8] the free configuration space C_f is reduced to a network of curves, and motion planning reduces to a search problem in this network. Examples of this approach are the visibility graph [6], Voronoi diagram [7] and silhouette [1] methods.

In the cell decomposition approach [2, 5, 11] the free configuration space C_f is decomposed into non-overlapping cells. A graph is constructed, where each node of the graph corresponds to a cell, and there is an edge between two nodes only if their corresponding cells are adjacent. A path from the start to the goal configuration is then constructed by first finding nodes in the cell graph that correspond to cells containing the start and goal configurations, and subsequently finding a path between these two nodes using a graph search algorithm.

In the potential field approach [3, 10] each configuration is assigned an obstacle potential as well as goal potential depending on the distance from obstacles and the goal, respectively. The potential of a configuration is a weighted sum of the obstacle potential and the goal potential. The path to the goal is taken by moving one step at a time in a direction (if it exists) that reduces the configuration potential, thus reaching the goal configuration which has least potential.

Each of the above mentioned approaches has its share of advantages and drawbacks. The skeleton and cell decomposition approaches, though tending to yield nearly optimal paths, become computationally expensive as the number of degrees of freedom of the robot (and hence the dimension of the configuration space) increases. With the potential field approach, which is typically easy to implement and also yields good paths, the problem is that the robot tends to get trapped in local minima of the potential field.

Recently Kavraki and Latombe [4] and Švestka and Overmars [9] proposed a new paradigm based on *randomization* to reduce computational effort. Their approach consists of two phases: a preprocessing phase followed by a path planning phase. In the preprocessing phase, a predetermined number of collision-free configurations are generated randomly in C_f and stored as nodes. Each time a node is generated a local planner is used to attempt to connect it to existing nodes that are closer, with respect to some metric, than a predetermined threshold value. An edge is inserted between nodes that the planner can successfully connect, to form a graph G, called the *configuration graph*. Thus edges in G represent the existence of a feasible path between pairs of nodes. In the path planning phase, one first attempts to connect the start configuration s and the goal configuration g to some nodes s' and g', respectively, in G. If this is successful one next attempts to find a path joining s' and g' in G.

We modify the randomized preprocessing approach, as proposed in [4, 9], by using a new heuristic to guide the initial generation of nodes. We term this the *cell division* approach. Our idea is to iteratively split the configuration space C into cells (i.e., cubes of the same dimension as C), examine each cell to determine if it is fully or partially occupied by obstacles, and use this information to decide if a node should be generated in that cell or if the cell should be split further. This heuristic tends to generate more nodes in "harder" regions of the configuration space, such as narrow corridors and near

A-1

obstacle edges, thus facilitating the movement of the robot in these regions. Experiments (simulated, of course) using cell division to move a rod and a disc in a plane amidst obstacles yielded positive results. Though the decomposition of the configuration space C into cells has, of course, been applied previously to plan motion directly, our contribution is to adapt it to generate configurations in C as a component of randomized preprocessing.

Next, we describe an entirely new approach to randomized preprocessing, the *ray shooting* approach. Ray shooting is dual to that of randomly generating configuration nodes: instead of randomly generating configurations, we generate straight rays (or tracks) in the configuration space C, along which the robot must travel. The ray segments which lie inside obstacles are infeasible and are filtered out. The remaining segments are considered as nodes and a local planner is used to attempt to connect nodes that are closer, with respect to some metric, than a predetermined threshold value. An edge is inserted between nodes that the planner successfully connects to form a graph G, called the *segment graph*. Path planning then consists of first attempting to connect the initial and final configurations of the robot to nodes in G and then finding a path in G joining these nodes. The major advantage of ray shooting seems to be that each node in the graph represents a segment that may transport the robot a relatively large distance through the configuration space. Experiments using ray shooting to move a rod on a plane amidst obstacles, even with minimal preprocessing, afforded encouraging results.

Our implementations were in C, on a DEC 5000/260 processor running at approximately 80MIPS. A graphical user interface, implemented in X windows, was used to draw scenes, specify query positions, and view results.

Section 2 gives a brief overview of the randomized preprocessing approach. In Section 3 we describe cell division, and in Section 4 we describe ray shooting. We conclude and discuss future work in Section 5.

2 Randomized Preprocessing Approach

As mentioned in the previous section, the randomized preprocessing approach [4, 9] consists of two phases: a preprocessing phase in which a configuration graph G representing the free configuration space C_f is constructed, followed by a path planning phase in which collision-free paths for the robot between the given initial and goal configuration pairs are found using G.

To give details, in the preprocessing phase, a predetermined number N of free configurations is generated randomly in C_f and stored as nodes of G. Each time a node (configuration) a is generated, we attempt to find a collision-free path for the robot from a to *some* nodes that already exist in G, using a *local planner*. (The local planner typically is an extremely fast but simple planner.) If the local planner is successful, then we say a is *connected* to these nodes.) An edge is inserted between nodes that the planner can successfully connect, to form the configuration graph G in which a path between two nodes represents the existence of a collision-free motion for the robot between the two corresponding configurations. (G may consist of more than one connected component, though.)

Subsequently, in the path planning phase, one first attempts to connect, possibly using the same local planner, the start configuration s and the goal configuration g to

some nodes s' and g', respectively, in G. If this is successful, one next attempts to find a path joining s' and g' in G. Of course, such a path exists if and only if s' and g' belong to the same connected component of G.

Since the local planner may not be very powerful, attempts to connect pairs of nodes that are "far apart" in C_f are likely to fail, and a number of different strategies have been proposed for selecting those nodes for which we attempt connection [9]. All use a distance function D among configurations and a constant *maxdist*, a threshold distance, for applying the local planner. Specifically, let

$$N(a) = \{ \text{ nodes } b : b \text{ exists in } G \text{ when } a \text{ is generated, } D(a,b) \leq maxdist \}.$$

Members of $N(a)$ are called *neighbors* of a. The strategy in which we attempt to connect a to *all* neighbors in $N(a)$ is called the *all-neighbors* method. In the *forest* method, on the other hand, we attempt to connect a to exactly one node in each connected component of G that has at least one node in $N(a)$. A strategy that lies between these two is the *nearest-k* method in which we attempt to connect a to at most k neighbors in $N(a)$, for the given k. Among these three strategies, the all-neighbors method tends to give shorter paths in the path planning phase than the other two methods. (Since the existence of an edge between two nodes indicates that the corresponding configurations are "close" to each other, it is usually the case that a path having fewer edges in G tends to represent a robot motion in the workspace that is physically shorter.) On the other hand, if the length of the paths discovered in the path planning phase is not a concern, the forest method gives just as much (transitive) connectivity among the nodes as the all-neighbors method does. The *heuristic loops* method of Overmars and Švestka [9] is an attempt to choose additional nodes for connection to get shorter paths in the path planning phase at moderate expense.

One way that we can fail in the path planning phase is that the nodes s' and g' to which we have connected the start and goal configurations s and g, respectively, do not belong to the same connected component of G. Some heuristics have been proposed to decrease this possibility. One suggested by Kavraki and Latombe [4] is to randomly generate extra nodes near those nodes in G that have a small degree. (The degree of a node is the number of edges incident on it.) Nodes having a small degree are considered to be in "difficult" regions of C_f, and thus additional paths (in G) through such regions might connect and thus merge two or more connected components, reducing the total number of connected components in G. Another heuristic is the *adaptive node adding* method of Overmars and Švestka [9], in which a node a when generated is not added to the graph if it is not considered useful (to save computational resources for processing other nodes that are, in fact, more useful), where a is *useful* if either (1) it is close to two different connected components (since a might bridge the two components), or (2) it is not close to any node in G (since a lies in an unexplored region).

Other heuristics include the *geometric node adding* method of Overmars and Švestka [9], where additional "important" configurations are generated in which the robot is near the edges and vertices of the obstacles.

Usually, the paths found in the path planning phase are jagged. They can be smoothed using the smoothing technique described in [9].

We implemented the randomized preprocessing approach using uniformly random generation of configurations and the forest edge-adding strategy, *without* any of the heuristics

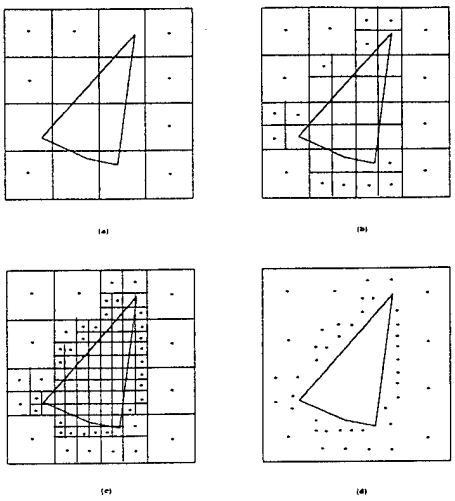

Figure 1: Node generation in cell division.

mentioned above to enhance the performance, which, for future reference, we term the KLOS approach.[1] (We describe the robots to which KLOS was applied and other implementation details in the next section.)

3 Cell Division

3.1 Overview

As described in the previous section, a critical component of randomized preprocessing is node (configuration) generation. The motivation underlying the cell division approach, as for geometric node adding and adaptive node adding, is to generate nodes in "harder" regions, based on a certain heuristic. Our objective is to capture the connectivity of the configuration space by adding more nodes near obstacle edges in crowded regions of the configuration space, and fewer in sparser regions where it is comparatively easier for the local planner to find a path joining two nodes.

Specifically, in the node generation phase of cell division approach, we first divide the configuration space C into cells of side length L. (The value of L, which is determined by the designer, is critical to performance and in the next paragraph we discuss factors that need to be taken into account when choosing L.) Then each cell is checked to determine if it is empty, or fully occupied, or partially occupied by obstacles. If the cell is empty a node is generated randomly in it. If the cell is fully occupied it is discarded. If the cell is partially occupied by obstacles then it is further divided into 2^d (d is the dimension of the configuration space) cells of side length $L/2$, and the process is repeated until either the side of each cell is less than some predetermined threshold value ϵ, or until the prespecified number N of nodes are generated (this is shown in Figure 1).

[1] Thus "KLOS" refers to an implementation of the most basic randomized preprocessing approach, and *not* the best possible strategy one might obtain using the heuristics of [4, 9].

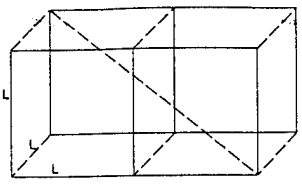

Figure 2: Maximum distance between two configurations in neighboring cells.

We next discuss certain factors influencing the choice of L. Recall from Section 2 that pairs of nodes within a distance of *maxdist* of each other are considered neighbors, and the local planner only attempts to connect some subset of such pairs (to give edges in G) in the preprocessing phase, depending on the edge-adding method used. With cell division, when the configuration space is of dimension d, two nodes in neighboring cells of side length L can be at most a distance of $(\sqrt{d+3})L$ apart (see Figure 2). Thus L should be chosen to be at most $\frac{maxdist}{\sqrt{d+3}}$, in order that the local planner does indeed attempt to connect nodes in adjacent cells and keep the number of components of G to a minimum. However, smaller L values result in a larger number of cells thereby increasing both the preprocessing time and the number of nodes generated in sparse regions (it should be noted although such nodes may not be useful in finding a path per se, they tend to optimize paths that are found in the path planning phase). A rule of thumb is to use larger L values for sparse scenes, and smaller values for more dense scenes.

3.2 Experiments

Robots:
We experimented with two kinds of robots that move on a plane:

1. Two movers carrying a rod of length 100 pixels at its endpoints.

2. Three movers carrying a disc of diameter 100 pixels; the movers are fixed with respect to the disc and are 120° apart from each other.

Scenes:
We experimented moving our robots in three different scenes, Scenes 1–3 shown in Figures 3–5. Each scene is a square of side 500 pixels containing obstacles that are *polygonal pools of water*, so that a mover *cannot* enter such an obstacle, but a rod or disc *can* straddle an obstacle (see Figure 6). Scene 1 contains a large number of small obstacles distributed fairly uniformly. In Scene 2 the free space is divided into two parts separated by many obstacles. We designed this scene mainly to test whether cell division that tends to generate a larger number of nodes in critical regions has an advantage over the uniformly random node generation of KLOS. Scene 3 is similar to Scene 2, except that the free space is divided into four parts by the obstacles.

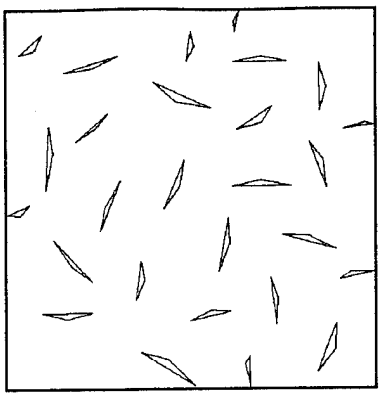

Figure 3: Scene 1.

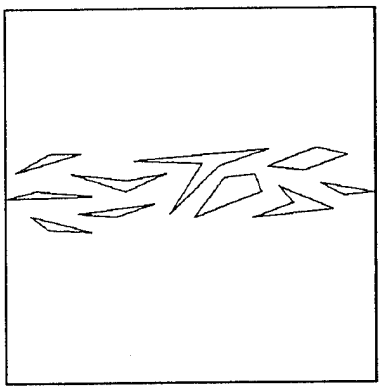

Figure 4: Scene 2.

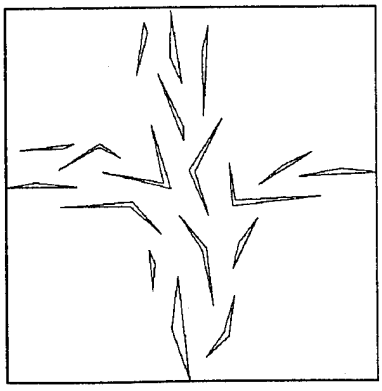

Figure 5: Scene 3.

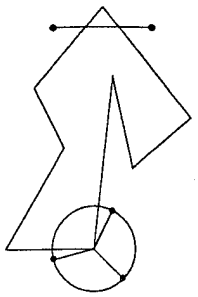

Figure 6: Valid configurations of a rod and a disc. The dots represent the movers.

3.3 Implementation Details

Rod:
The configuration space is three-dimensional, each configuration being a 3-tuple (x, y, θ), where (x, y) are the co-ordinates of a distinguished endpoint of the rod, and θ is its orientation. Because of the particularly simple nature of our robot, when implementing cell division, we only divide the planar scene into square cells and examine if a cell intersects an obstacle. If a cell is found to be obstacle-free, then a configuration of the rod having the distinguished endpoint in that cell is generated randomly. The metric D measures the distance between two configurations (x_1, y_1, θ_1) and (x_2, y_2, θ_2) as the Euclidean distance $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ between their distinguished endpoints. We used $maxdist = 50$ pixels for attempting connection using our local planner.

Our local planner tries two heuristics in order. The second one is tried only if the first one fails. If both the heuristics fail, then no edge is added between the configurations.

1. The local planner tries to move from configuration (x_1, y_1, θ_1) to configuration (x_2, y_2, θ_2) by first moving the rod translationally, parallel to itself, such that the distinguished endpoint at (x_1, y_1) reaches (x_2, y_2). Then the rod is rotated about the distinguished endpoint through an angle of $\theta_1 - \theta_2$ (see Figure 7(a)). It is checked that endpoints do not enter obstacles during either translational or rotational movement.

2. This case is symmetric to the previous one, except that the local planner attempts rotation before translation (see Figure 7(b)).

Disc:
The configuration space again is three-dimensional, each configuration being a 3-tuple (x, y, θ), where (x, y) are the co-ordinates of the center of the disc, and θ is the orientation of the line joining the center and a distinguished mover. As in the case of the rod, when implementing cell division, we only divide the planar scene into square cells and examine if a cell intersects an obstacle. If a cell is found to be obstacle-free, then a configuration of the disc having the center in that cell is generated randomly. The metric D measures the distance between two configurations (x_1, y_1, θ_1) and (x_2, y_2, θ_2) as the Euclidean distance $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ between the positions of the center. We used $maxdist = 50$ pixels for attempting connection using our local planner.

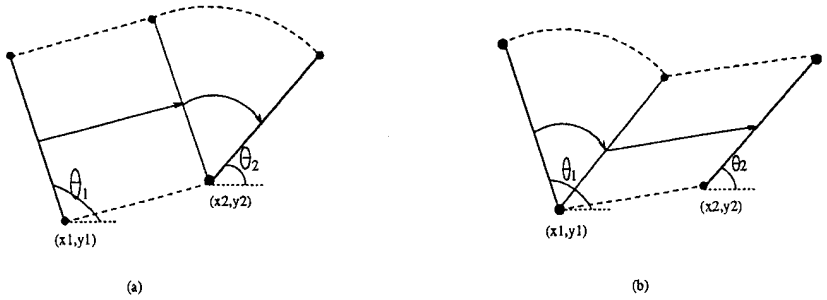

Figure 7: Movement of rod.

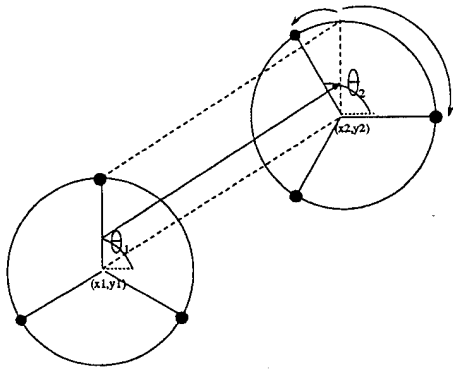

Figure 8: Movement of disc.

Our local planner tries to move from configuration (x_1, y_1, θ_1) to configuration (x_2, y_2, θ_2) by first moving the disc translationally, without changing the orientation of any mover, until the center reaches (x_2, y_2). Then the disc is rotated about its center until the movers occupy the desired positions. (See Figure 8.) This rotational movement is attempted in either direction, if necessary, to avoid collision of the movers with obstacles.

We apply the forest edge-adding strategy when creating configuration graphs for both rod and disc.

When implementing KLOS for both rod and disc we used the same local planners as for cell division.

3.4 Results

3.4.1 Data

For each scene, we generated 1000 source-goal pairs of valid query configurations. (Not all of the source-goal pairs, however, may have a collision-free motion of the movers.) If in the path planning phase we can find a path for a given source-goal pair, we say the query is answered *successfully*.

For both the KLOS and cell division approach we tabulated:

1. N: Number of nodes in the configuration graph.

2. NE: Number of edges in the configuration graph.

3. C: Number of connected components in the configuration graph.

4. T: CPU time in seconds for preprocessing.

5. SQ: Number of queries answered successfully.

6. NC: Number of queries that could not be successfully answered because the local planner failed to connect either the start configuration s or the goal configuration g to the configuration graph.

7. NP: Number of queries that could not be successfully answered because the nodes s' and g' of the configuration graph to which s and g were connected, respectively, belonged to different components.

8. SP: Average length of the paths of the robot (rod or disc) for queries answered successfully, where the length of a path is the sum of the distances between successive configurations along the path.

9. TPQ: Average CPU time in seconds to answer a query.

An example of data we collected is shown in Table 1. The table contains the results we obtained for the case of the rod in Scene 3, using the cell division approach, where the *maxdist* is 50 pixels. In this version we do not present further raw data, rather we display charts in Appendix A that plot SQ versus N for each of the three scenes, for cell division and KLOS, for both the rod and the disc. Charts plotting SQ versus preprocessing time T for cell division and KLOS (as well as for ray shooting explained later for the case of the rod), are also given in Appendix A.

3.4.2 Analysis

We were expecting that, since dividing the configuration space into cells requires extra computation, preprocessing time T for cell division would far exceed that of KLOS. Interestingly, we found this not to be the case. For all three scenes, cell division, in fact, uses less preprocessing time than KLOS for all values of N. We believe that T tends to be less for cell division for the following reason:
In both KLOS and cell division, as new configurations are generated they are checked for validity, and rejected if invalid. This process continues until the number N of valid configurations is reached. However, with cell division it seems the probability that a generated configuration is invalid is less because no attempts are made to generate a configuration (more precisely, to generate the distinguished endpoint of the rod or the center of the disc) in a cell that is not obstacle-free.

Let us analyze the results for the rod carefully (Figures 10–12, 13–15). With one exception that we will mention shortly, cell division outperformed KLOS in the number of queries SQ answered successfully for all scenes, for small and moderately large values of N.

For Scene 1 (Figures 10 and 13), cell division solves over 800 queries with N=375 and preprocessing time of T=1.27 seconds, whereas KLOS starts to solve nearly 800 queries

Table 1: Results of cell division for the rod in Scene 3 with $maxdist = 50$ pixels.

N	NE	C	T	SQ	NC	NP	SP	TPQ
125	110	15	0.25	82	823	95	277.08	0.01
250	203	47	0.63	156	105	739	288.84	0.04
375	322	53	1.05	179	40	781	291.39	0.08
500	424	76	2.13	191	19	790	304.16	0.11
625	551	74	3.27	257	16	727	377.73	0.16
750	680	70	4.85	821	16	163	1035.36	0.20
875	807	68	6.64	808	14	178	1038.50	0.26
1000	929	71	8.63	813	10	177	1039.14	0.33
1125	1061	64	10.51	895	9	96	1066.44	0.39
1250	1178	72	13.47	892	8	100	1071.22	0.47
1375	1301	74	16.13	886	7	107	1071.64	0.57
1500	1434	66	19.95	920	7	73	1065.27	0.66
1625	1549	76	22.59	912	7	81	1068.45	0.77
1750	1661	89	27.10	900	7	93	1070.69	0.88
1875	1791	84	30.50	917	7	76	1069.28	1.01
2000	1921	79	33.20	923	7	70	1073.51	1.13

only after N reaches 875 and T is over 22 seconds. Beyond these points, both approaches solve nearly the same number, about 900, of queries. The reason for this seems to be that, for smaller N, cell division is at an advantage as it tends to generate a larger number of nodes in critical regions. KLOS catches up when N becomes large because, in such cases, even "uninformed" random node generation covers critical regions.

We observe a similar phenomenon for the other two scenes. For Scene 2 (Figure 11 and 14), cell division seems to "break through" the wall of obstacles and solve nearly 950 queries with N=625 and preprocessing time of T=1.61 seconds, whereas the same happens to KLOS only after N=1000 and T=20.65 seconds. Again, the performance of the two approaches in terms of SQ become nearly identical beyond these. For Scene 3 (Figures 12 and 15), the observation given above still applies, in that cell division solves over 800 queries after N=750 and T=4.85 seconds, whereas it is not until N=1500 and T=79.78 seconds before KLOS solves over 800 queries. One exception is the case N=625 when cell division solves about 250 queries while KLOS solves over 600 queries. However, generating 625 nodes in KLOS requires over 12 seconds, while 12 seconds is sufficient for cell division to generate over 1125 nodes with which it can solve nearly 900 queries.

The average time TPQ it took to answer queries was almost the same for both cell division and KLOS. This seems because graphs for both are of similar sizes. The average path length SP is typically slightly more with cell division than KLOS as there are fewer nodes available in sparse regions that help optimize path length.

The results for the disc (Figures 16–18, 19–21) largely follow a similar pattern for Scenes 2 and 3, but not for Scene 1. For Scene 1 (Figures 16 and 19), cell division and KLOS solve over 900 queries with N=500 and T=2.59 seconds, and with N=375 and T=2.81 seconds, respectively. So KLOS outperformed cell division with a small margin for smaller values of N. One might speculate that Scene 1 happens to be an "easy" scene for the disc of this size, and thus the difference between the two approaches turned out

to be insignificant. For Scene 2 (Figures 17 and 20), cell division solves over 900 queries with N=750 and T=5.15 seconds, whereas KLOS requires N=1625 and T=47.64 seconds to do the same. For Scene 3 (Figures 18 and 21), cell division outperforms KLOS up to N=875, but the difference (in terms of SQ) is small. However, cell division uses only about 12 seconds of preprocessing time to solve over 900 queries, while KLOS requires over 36 seconds.

In summary, cell division seems preferable when either preprocessing time is at a premium or the scene is complicated with many critical regions that the planner must break through.

4 Ray Shooting

4.1 Overview

Ray shooting is also a two-phase approach with preprocessing followed by path planning. The main motivation for this approach is that it may be possible to move from one configuration c to another c', that is far from c, along relatively few straight line segments in the configuration space, whereas, with any strategy based on random node generation, it is likely that only after a large number of nodes is generated that c and c' would belong to the same component. Therefore, it seems intuitive that random generation of straight rays (that represent possible trajectories of the robot) would tend to rapidly "cover and connect" the configuration space.

Given this motivation, in the preprocessing phase of ray shooting, we randomly generate rays, instead of nodes, in the configuration space. The segments of rays that lie inside obstacles are infeasible, and filtered out. The remaining segments are treated as nodes of a graph G, the segment graph. Whenever two segments ℓ and m are "sufficiently" close with respect to a given metric, a local planner (again, preferably simple and fast) attempts to connect some configuration in ℓ to some configuration in m. If the local planner succeeds, then an edge is added in G between the nodes representing ℓ and m.

Path planning then consists of first attempting to connect the initial and final configurations of the robot to nodes in G, and then finding a path in G joining these nodes.

4.2 Experiments

We applied the ray shooting approach to a robot consisting of a rod with movers at both endpoints in the three scenes as described in Section 3.2.

4.3 Implementation Details

Again, due to the simple nature of our robot, instead of generating rays in the three-dimensional configuration space we generate rays directly on the planar scene. The segments of these rays that lie inside obstacles are filtered out and the remaining segments form nodes of the segment graph G.

Our local planner then attempts to move the rod from segment ℓ to segment m of G for every pair of *intersecting* segments ℓ and m. If the local planner succeeds then an

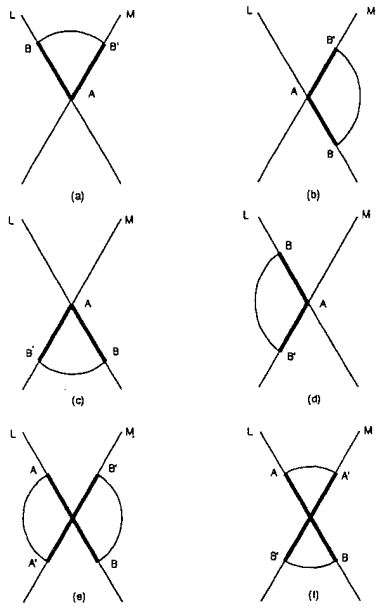

Figure 9: Local planner for the rod in ray shooting.

edge is added in G between the nodes representing ℓ and m. The local planner tries the following heuristics in order when attempting to move the rod from ℓ to m.

- The rod, say AB, moves on and along segment ℓ until one endpoint, say A, reaches the intersection of ℓ and m, and then the rod is rotated about A until the other endpoint lies on m (if necessary, to avoid collision, an attempt is made to rotate in either possible direction). See Figure 9(a)–(d).
- The rod again moves on and along line ℓ until the center, say O, of the rod reaches the intersection of ℓ and m, and then the rod is rotated about O until it lies on m (if necessary, to avoid collision, an attempt is made to rotate in either possible direction). See Figure 9(e)(f).

In the path planning phase, we must first attempt to connect the start s and goal g to nodes s' and g', respectively, of G. This is done as follows:

First a ray r is "fired" along the rod, in either direction, and the intersections, if any, of these rays with segments of G are found. If such an intersection does, in fact, exist the local planner is used to attempt to move the rod from r to the intersecting segment ℓ. If this fails, the rod is rotated by an angle of $45°$, and the process is repeated until the rod has turned $180°$.

4.4 Results

4.4.1 Data

In our experiments with the ray shooting approach, we again generated 1000 source-goal pairs of valid query configurations. We tabulated the following:

Table 2: Results of ray shooting for the rod in Scene 3.

NR	N	NE	C	T	SQ	NC	NP	TPQ
5	11	1	4	0.06	269	512	201	0.01
10	20	18	5	0.11	324	361	315	0.03
20	39	60	6	0.28	328	350	322	0.05
30	58	152	9	0.65	330	344	326	0.11
40	77	240	9	0.77	338	336	326	0.11
50	94	377	8	1.30	395	335	270	0.15
100	181	1587	12	3.42	398	325	277	0.28
150	275	3688	11	11.80	400	323	277	0.60
200	357	6222	14	11.99	670	323	7	0.57
250	447	9885	17	18.44	673	321	6	0.74
300	534	14255	17	25.56	675	321	4	0.90
350	626	19729	19	34.96	675	321	4	1.08
400	710	25150	15	43.45	677	321	2	1.08
450	800	31956	16	54.64	677	321	2	1.40
500	889	39529	17	67.49	677	321	2	1.59

1. NR: Number of random rays generated.
2. N: Number of nodes in the segment graph (i.e., segments remaining after filtering).
3. NE: Number of edges in the segment graph.
4. C: Number of connected components in the segment graph.
5. T: CPU time in seconds for preprocessing.
6. SQ: Number of queries answered successfully.
7. NC: Number of queries that could not be successfully answered because the local planner failed to connect either the start configuration s or the goal configuration g to the segment graph.
8. NP: Number of queries that could not be successfully answered because the nodes s' and g' of the segment graph to which s and g were connected, respectively, belonged to different components.
9. TPQ: Average CPU time in seconds to answer a query.

An example of data we collected is shown in Table 2. The table contains the results we obtained for the case of the rod in Scene 3, using the ray shooting approach. In this version we do not present further raw data, rather we display charts (in Appendix A) that plot SQ versus T for each of the three scenes, for cell division, KLOS and ray shooting. (Since N represents the number of segments in ray shooting while it represents the number of individual configurations in cell division and KLOS, we found it more meaningful to compare the three methods using SQ versus T rather than SQ versus N.)

4.4.2 Analysis

As indicated in Figures 13–15 that plot SQ versus preprocessing time T for all three approaches we implemented, ray shooting tends to successfully answer more queries than the other two approaches when T is very small. This seems to justify motivations underlying the ray shooting approach. Specifically, for Scene 1, ray shooting solves more than 300 queries with T as little as 0.07 seconds and over 650 queries with T=0.37 (versus 173 queries at 0.35 seconds for cell division, and 100 queries at 0.5 seconds for KLOS). For Scene 2, ray shooting solves 555 queries with T=0.02, while cell division and KLOS solve about 400 queries with T=0.16, and about 150 queries with T=0.31, respectively. For Scene 3, ray shooting solves 324 queries with T=0.11, while cell division and KLOS solve about 80 queries with T=0.25, and about 45 queries with T=0.50, respectively. Average time TQP to answer a query was about the same as that for cell division and KLOS.

Unfortunately, however, the performance of ray shooting seems to level off rather quickly as the number NR of randomly generated rays increases. (Charts plotting SQ versus NR are not included in this paper, but the reader can see the level off in Figures 13–15.) Specifically, SQ remains slightly over 700 beyond NR=100 for Scene 1. For Scene 2, SQ is as large as 555 even for NR=5 but it remains essentially the same even for NR=500. For Scene 3, SQ remains about 670 beyond NR=200. This is not very surprising for Scenes 2 and 3, since ray shooting does not seem to be particularly suited for breaking thorough the barriers of obstacles. For Scene 1, we expected the performance of ray shooting to continue to improve as NR becomes larger. We do not as yet fully understand the reason for this, but suspect it relates to the length of the rod relative to the complexity of the scene (i.e., SQ should level off at a higher value if the rod is shorter).

In summary, ray shooting seems a promising new approach that seems to yield an efficient graph for the path planning phase with minimal computational effort in the preprocessing phase.

5 Conclusions and Future Work

We have proposed a new heuristic, cell division, for the configuration generation phase of randomized preprocessing, as well as an entirely new approach to randomized preprocessing, ray shooting. Experimental results for both seem promising and support expectations.

Future work includes:

1. Improving local planners to enhance the performance of both cell division and ray shooting in moving the rod and disc.

2. Applying both cell division and ray shooting to other robots.

3. In particular, experimenting with ray shooting in higher-dimensional configuration spaces, where we must deal with *skew* lines, and surrender the advantage of intersecting line segments that we had when moving a rod on a plane.

4. A hybrid approach combining cell division and ray shooting, motivated by the observation that cell division should be effective in denser regions, while ray shooting is likely to be effective in sparser regions where ray segments tend to be longer.

5. Most importantly, theoretical verification of our experimental results.

References

[1] J.F. Canny, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[2] E.B. Feinberg and C.H. Papadimitriou, "Finding Feasible Paths for a Two-Point Body," *Journal of Algorithms*, Vol. 10, pp. 109–119, 1989.

[3] Y.K. Hwang and N. Ahuja, "Gross Motion Planning - A Survey," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 219–291, 1992.

[4] L. Kavraki and J.-C. Latombe, "Randomized Preprocessing of Configuration Space for Fast Path Planning," Technical Report STAN-CS-93-1490, Dept. of Comp. Sci., Stanford Univ., Stanford, CA, September 1993.

[5] Y. Ke and J. O'Rourke, "Moving a Ladder in Three Dimensions: Upper and Lower Bounds," in *Proceedings of the Third Annual Symposium on Computational Geometry*, ACM, pp. 136–145, 1987.

[6] T. Lozano-Perez and M.A. Wesley, "An Algorithm For Planning Collision Free Paths Among Polyhedral Obstacles," *Communication of the ACM*, Vol. 22, pp. 560–570, 1979.

[7] C. O'Dúnlaing and C.K. Yap, "A "Retraction" Method for Planning the Motion of a Disc," *Journal of Algorithms*, Vol. 6, pp. 104–111, 1985.

[8] C. O'Dúnlaing, M. Sharir, and C.K. Yap, "Generalized Voronoi Diagrams for a Ladder. I. Topological Considerations," *Comm. Pure Appl. Math.*, Vol. 39, pp. 423–483, 1986.

[9] M.H. Overmars and P. Švestka, "A Probabilistic Learning Approach to Motion Planning," Technical Report UU-CS-1994-03, Comp. Sci., Utrecht Univ., the Netherlands, January 1994.

[10] S. Ratering and M. Gini, "Robot Navigation in a Known Environment with Unknown Moving Obstacles," in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 25–30, 1993.

[11] J. T. Shwartz and M. Sharir, "On the Piano Movers' Problem I. The Case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Comm. Pure Appl. Math.*, Vol. 36, pp. 345–398, 1983.

A Simulation Results

Figures 10–12 show charts plotting SQ versus N for the rod in Scenes 1–3, respectively, by cell division and KLOS.

Figures 13–15 show charts plotting SQ versus T for the rod, by cell division, KLOS and ray shooting.

Figures 16–18 show charts plotting SQ versus N for the disc, by cell division and KLOS.

Figures 19–21 show charts plotting SQ versus T for the disc, by cell division and KLOS.

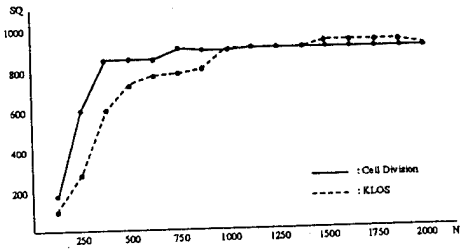

Figure 10: Rod, SQ vs. N, Scene 1.

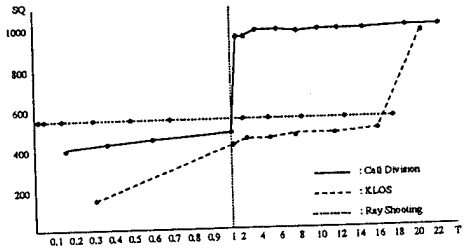

Figure 14: Rod, SQ vs. T, Scene 2.

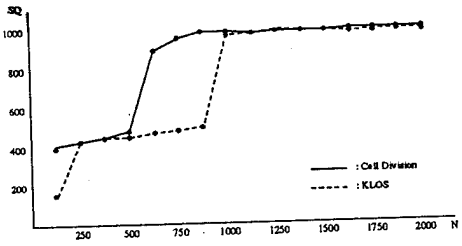

Figure 11: Rod, SQ vs. N, Scene 2.

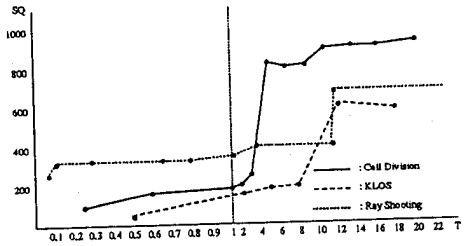

Figure 15: Rod, SQ vs. T, Scene 3.

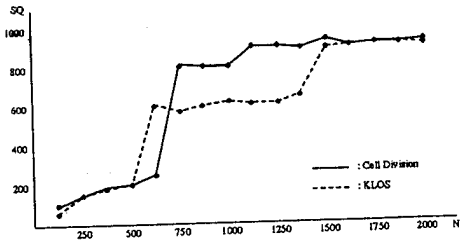

Figure 12: Rod, SQ vs. N, Scene 3.

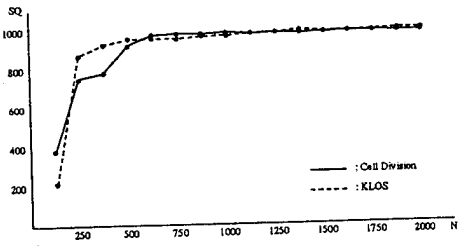

Figure 16: Disc, SQ vs. N, Scene 1.

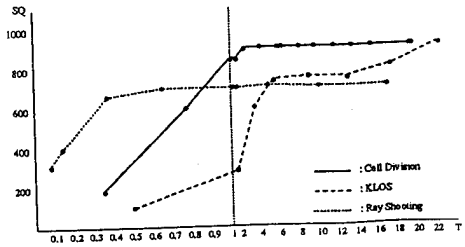

Figure 13: Rod, SQ vs. T, Scene 1.

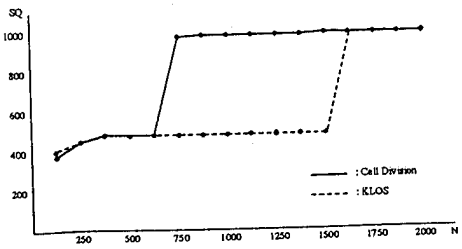

Figure 17: Disc, SQ vs. N, Scene 2.

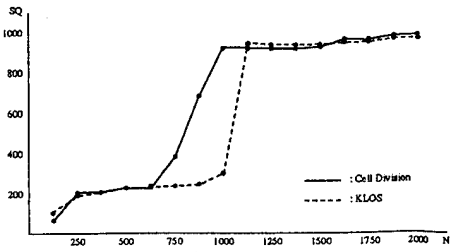

Figure 18: Disc, SQ vs. N, Scene 3.

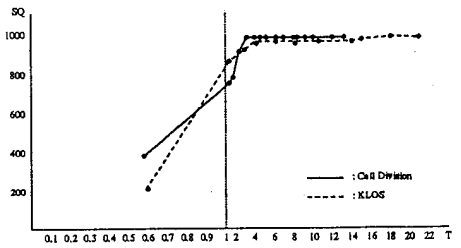

Figure 19: Disc, SQ vs. T, Scene 1.

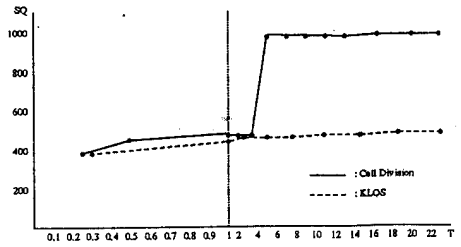

Figure 20: Disc, SQ vs. T, Scene 2.

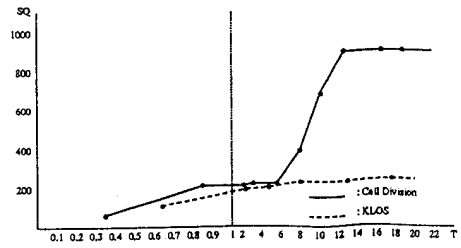

Figure 21: Disc, SQ vs. T, Scene 3.